Best Ways to Connect Local AI to Notion in 2026

March 11, 2026 · by Mark Bartlett

Download this guide as PDF

> **Quick Answer:** The fastest path is Open WebUI + Notion's official MCP server — there's a working tutorial and it connects to your local Ollama models. The most private path is a RAG pipeline that exports your Notion pages, embeds them locally with ChromaDB, and queries them with Ollama (fully offline after the initial sync). MCP is the hot approach right now, but Ollama doesn't support MCP natively — you need a bridge tool like mcphost or Open WebUI's MCPO adapter. For automation workflows (summarize new pages, sync databases), n8n with Ollama is the mature option. Pair any of these with Qwen 3.5 9B for general queries or Qwen 3.5 35B-A3B for complex reasoning. None of these approaches are one-click easy yet. Budget 30-60 minutes for initial setup.

Notion users keep asking the same question on Reddit: can I search, summarize, and generate content in my Notion workspace using a local model, with nothing leaving my machine?

The answer is yes, with caveats. Four approaches work today, each with different tradeoffs between privacy and setup pain. None of them are one-click. All of them require a terminal.

I tested each one. Some are genuinely useful. Others are more "technically possible" than "actually pleasant."

The 4 approaches at a glance

Approach	Privacy	Setup	Read Notion	Write to Notion	Best for
Open WebUI + Notion MCP	Medium (API calls at query time)	3/5	Yes	Yes	Chat-style queries against your workspace
mcphost + Notion MCP + Ollama	Medium (same API dependency)	4/5	Yes	Yes	CLI users, power users
RAG pipeline (export + embed locally)	High (fully offline after sync)	4/5	Yes	No	Maximum privacy, search-heavy use

Approach	Privacy	Setup	Read Notion	Write to Notion	Best for
n8n + Ollama	Medium (API calls to Notion)	3/5	Yes	Yes	Automated workflows, event triggers

One thing to understand upfront: every approach except RAG requires your machine to talk to Notion's API at query time. Your LLM runs locally and your prompts never leave your machine, but Notion page content gets fetched over the network. True air-gapped privacy requires the RAG approach.

Approach 1: Open WebUI + Notion MCP server

The fastest path to something that actually works. Open WebUI has an official tutorial for Notion MCP integration, and it connects to any Ollama model.

What it does

Open WebUI connects to Notion's MCP (Model Context Protocol) server, which gives your local model 22 tools for searching, reading, creating, and editing Notion pages and databases. You chat in Open WebUI's interface and the model calls Notion's API on your behalf.

Setup steps

1. Create a Notion integration and grab your API key
2. Share the pages you want accessible with your integration (Notion gives zero permissions by default)
3. Set up the MCP connection in Open WebUI — two options:
   - **Streamable HTTP** (connects to `mcp.notion.com/mcp` via OAuth — easier but requires Notion cloud)
   - **Self-hosted via MCPO bridge** (Docker container running the MCP server locally — more private)

For the self-hosted option:

```
docker run -p 8010:8000 \
  -e OPENAPI_API_KEY=ntn-your-notion-key \
  ghcr.io/open-webui/mcpo:latest \
  --api-key "your-mcpo-key" \
  -- npx -y @notionhq/notion-mcp-server
```

Then add the MCP connection in Open WebUI's admin settings and point it at `http://localhost:8010`.

Best model pairing

Qwen 3.5 9B handles most Notion queries well — summarizing pages, searching databases, drafting content. For complex multi-step operations (query a database, filter results, create a new page with the summary), step up to Qwen 3.5 35B-A3B or Qwen 3.5 30B.

The gotchas

- OAuth sessions expire after inactivity or container restarts. You'll need to re-authenticate periodically.
- Domain mismatches during OAuth cause CSRF failures. Make sure your `WEBUI_SECRET_KEY` is persistent across restarts.
- Smaller models (7-9B) sometimes botch complex multi-tool MCP calls. They'll call the wrong tool or pass malformed arguments. I had Qwen 3.5 9B try to search a database by passing the page title as the database ID. 30B+ models are more reliable here.
- The stdio MCP transport isn't natively supported in Open WebUI — you need the MCPO Docker bridge for the self-hosted path.

Setup complexity: 3/5

Privacy level: Medium

Your prompts and model responses stay local. But every time you ask about a Notion page, the MCP server fetches it from Notion's API. Your Notion data is already on Notion's servers, so this doesn't make things worse, but it's not air-gapped either.

Approach 2: mcphost + Notion MCP server + Ollama

For CLI users who want a lighter setup without Open WebUI's full interface.

What it does

mcphost (1,600+ stars) is a Go CLI tool that connects any Ollama model to any MCP server. You configure it with a JSON file pointing at the Notion MCP server, pick your model, and chat in the terminal.

Setup steps

1. Install mcphost:

```
go install github.com/mark3labs/mcphost@latest
```

1. Create a config file (`~/.mcphost/config.json`):

```
{
  "mcpServers": {
    "notion": {
      "command": "npx",
      "args": ["-y", "@notionhq/notion-mcp-server"],
      "env": {
        "OPENAPI_MCP_HEADERS": "{\"Authorization\": \"Bearer ntn-your-notion-key\", \"Notion-Ve
      }
    }
  }
}
```

1. Run it:

```
mcphost --model ollama:qwen3.5:9b
```

You get an interactive terminal where you can ask questions about your Notion workspace.

The catch: function-calling models only

MCP requires the model to output structured tool calls. Only models with function-calling support work. This rules out reasoning-only models like DeepSeek-R1 in pure thinking mode.

Models that work well with MCP:

Model	Size	Function calling	Notion MCP quality
Qwen 3.5 9B	9B	Yes	Good for simple queries
Qwen 3.5 30B	30B (MoE, ~8B active)	Yes	Good for multi-step
Qwen 3.5 35B-A3B	35B (MoE, 3B active)	Yes	Decent, sometimes drops tools
Llama 4 Scout	109B (MoE, 17B active)	Yes	Best quality, needs 32GB+

Setup complexity: 4/5

Requires Go installed, comfort with CLI tools, and understanding of MCP configuration. Not hard, but not beginner-friendly.

Privacy level: Medium

Same as Approach 1 — local model, but API calls to Notion for page content.

Approach 3: RAG pipeline (fully offline after sync)

The nuclear privacy option. Export your Notion pages, embed them locally, query the embeddings with Ollama. After the initial sync, your machine never talks to Notion again.

What it does

1. Pull Notion pages via the API using a sync script
2. Chunk the pages into smaller pieces
3. Embed them locally with a sentence-transformer model
4. Store the embeddings in ChromaDB (or FAISS, or Qdrant)
5. Query with Ollama — the retriever finds relevant chunks, the LLM generates an answer

Architecture

```
Notion API → sync script → markdown files → chunker → embedder → ChromaDB
                                                                      ↓
                                                Ollama ← retrieved chunks ← query
```

Setup steps (Python)

```
pip install chromadb langchain langchain-community sentence-transformers ollama
```

```
from langchain_community.document_loaders import NotionDBLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import Chroma
from langchain_community.embeddings import HuggingFaceEmbeddings

# 1. Load pages from Notion
loader = NotionDBLoader(
    integration_token="ntn-your-key",
    database_id="your-database-id"
)
docs = loader.load()

# 2. Chunk
splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
chunks = splitter.split_documents(docs)

# 3. Embed and store locally
embeddings = HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
vectorstore = Chroma.from_documents(chunks, embeddings, persist_directory="./notion-db")

# 4. Query
results = vectorstore.similarity_search("What's our Q1 roadmap?", k=5)
context = "\n\n".join([r.page_content for r in results])

# 5. Send to Ollama
import ollama
response = ollama.chat(model="qwen3.5:9b", messages=[
    {"role": "system", "content": f"Answer based on this context:\n{context}"},
    {"role": "user", "content": "What's our Q1 roadmap?"}
])
print(response["message"]["content"])
```

Performance

People report 500+ page workspace syncs in under 90 seconds on an M1 Mac. Query response runs 1.2 to 2.8 seconds end-to-end. Works on machines with 16GB RAM.

The gotchas

- Read-only. RAG lets you search and summarize your Notion content, but you can't create or edit pages. You'd need a separate write pipeline for that.
- Stale data. Your local index is a snapshot. Re-sync periodically (cron job every hour, or on-demand) to pick up new pages and edits.
- Chunking matters. The default 1000-token chunks work for most content, but tables and databases lose their structure when chunked naively. You may need custom chunking logic for structured data.
- Most Notion RAG repos on GitHub are OpenAI-only. You'll need to swap the embedding model and LLM provider to local alternatives. The code above already does this.

For more on RAG pipelines, see our RAG guide for local AI.

Setup complexity: 4/5

Requires Python, several libraries, and some understanding of embeddings and vector databases. The code is straightforward. Debugging chunk quality and retrieval accuracy is where you'll spend your time.

Privacy level: High

After the initial sync from Notion's API, everything runs locally. Your queries, embeddings, and model responses never leave your machine. If you export Notion as markdown files instead of using the API, you can skip the API entirely.

Approach 4: n8n + Ollama (automated workflows)

Different philosophy from the other approaches. Instead of interactive chat, you build automated workflows that trigger on events and use a local model for processing.

What it does

n8n (178,000+ stars) is a self-hosted workflow automation tool with native Notion and Ollama integrations. You build visual workflows like:

- When a new page is added to a Notion database → summarize it with Ollama → post the summary to Slack

- Every morning → query a Notion database for overdue tasks → generate a digest with Ollama → email it to you
- When a Notion page is tagged “needs review” → send the content to Ollama for editing suggestions → add the suggestions as a comment

Setup steps

```
# Run n8n locally with Docker
docker run -it --rm \
  -p 5678:5678 \
  -v n8n_data:/home/node/.n8n \
  n8nio/n8n
```

Then open `http://localhost:5678`, add your Notion integration credentials, configure the Ollama node to point at `http://host.docker.internal:11434` (if Ollama runs on the host), and build your workflow visually.

n8n has pre-built nodes for Notion (read, create, update, search) and AI/LLM nodes that support Ollama as a backend. No coding required for basic workflows.

Best for

Event-driven automation where you want Notion and local AI to work together without you sitting in front of a chat window. Think cron jobs, not conversations.

The gotchas

- n8n is a separate service to run and maintain. It needs Docker and eats a few hundred MB of RAM.
- The Ollama node in n8n requires your Ollama instance to be network-accessible. If Docker networking confuses you, this is where you’ll get stuck. See our Ollama troubleshooting guide for common fixes.
- Complex LLM workflows in n8n can be flaky. If the model outputs unexpected JSON or the Notion API rate-limits you (180 requests/min general, 30/min for searches), the workflow fails and you need to debug it in n8n’s execution log.

Setup complexity: 3/5

The visual workflow builder is friendly. The Docker networking and API credential setup takes some patience.

Privacy level: Medium

n8n runs locally. Ollama runs locally. But the Notion nodes call Notion’s API, so page content flows over the network. Same privacy profile as the MCP approaches.

Approach 5: skip Notion entirely

If your main reason for wanting local AI in Notion is privacy, consider whether you need Notion at all. Local-first alternatives with built-in AI support:

Obsidian + local LLM — Obsidian stores everything as plain markdown files on your disk. Several community plugins (Smart Connections, Local GPT) connect directly to Ollama. No API, no sync, no network calls. Your notes and your model on the same machine. See our Mac guide for running Ollama on Apple Silicon.

SiYuan (41,800 stars) — self-hosted knowledge management with built-in Ollama and DeepSeek support. Block-based editor similar to Notion, but everything stays on your machine. Has its own AI features baked in.

Logseq — open-source, local-first knowledge management with a plugin ecosystem. Not as polished as Notion but fully offline-capable.

The tradeoff: you lose Notion’s collaboration features and its polished UI. If you work with a team that lives in Notion, these aren’t practical. If you’re a solo user willing to migrate, the privacy gains are real.

Which model to run

For all approaches, here’s what to pair with your Notion integration:

Use case	Model	VRAM needed	Why
	Qwen 3.5 9B	6-8 GB	

Use case	Model	VRAM needed	Why
Simple queries (search, summarize)			Fast, good function calling, handles most Notion tasks
Complex multi-step operations	Qwen 3.5 30B	18-20 GB	Better at chaining tool calls without dropping context
Budget/low VRAM	Qwen 3.5 35B-A3B	3-4 GB	MoE with only 3B active, runs on 8GB cards
Best quality	Llama 4 Scout	32+ GB	Largest context, best reasoning, but needs serious hardware

If you're on a Mac, Qwen 3.5 9B on Ollama with MLX gives you good speed with minimal setup. On an 8GB VRAM GPU, Qwen 3.5 9B at Q4 fits comfortably.

Common gotchas across all approaches

Notion permissions are opt-in. When you create a Notion integration, it has access to nothing. You must manually share each page or database with the integration. If your model says "I can't find any pages," this is almost certainly why.

Rate limits will bite you. Notion's API allows 180 requests per minute general and 30 per minute for search. If you're syncing a large workspace or running frequent automated queries, you'll need backoff logic.

MCP requires function-calling models. If you're using approaches 1 or 2, your model must support structured tool output. Reasoning-only models and older models without tool-call support won't work. Qwen 3.5 and Llama 4 handle this well.

Notion may sunset local STDIO MCP. Their team is prioritizing the remote/hosted MCP server at `mcp.notion.com`. The local STDIO transport (`npx @notionhq/notion-mcp-server`) still works but issues and PRs on the repo are "not actively monitored." Worth keeping in mind.

The bottom line

For most people, start with Open WebUI + Notion MCP (Approach 1). Best documentation, works with any Ollama model, read and write access.

If privacy is the whole point and you only need search and summaries, the RAG pipeline (Approach 3) is the only option that works fully offline after the initial sync.

For automation without sitting in a chat window, n8n (Approach 4). For terminal-only people, mcphost (Approach 2).

None of these are as polished as Notion's built-in AI, which costs $10/user/month. But your data stays on your machine, you pick your model, and there's no per-query cost after setup.

Source: https://insiderllm.com/guides/notion-local-ai-integration/